25th Annual
Software Engineering Workshop
November 28-30, 2000 at NASA/GSFC

Understanding Software for Project Management

Presented by: **P. A. "Trisha" Jansma**

Manager, CSMISS IT Workforce Enrichment Element

Jet Propulsion Laboratory, California Institute of Technology

JPL CSMISS Center for Space Mission Information and Software Systems ITWE

U. S. Government Sponsorship Acknowledged under NAS7-1260 and NAS7-1407.

Course Goals

- To provide a general awareness of software management issues for Project Managers, or others from a non-software background, who want to gain some familiarity with software issues that may affect the success of their projects.

- To help Project Managers know how to manage their software development activities better so that they can use their scarce Information Technology (IT) workers or "techies" more efficiently.

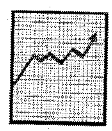
"Tech-Project Inefficiencies Found in Corporate Study" by Rachel Emma Silverman, Wall Street Journal, Nov. 14, 2000

Topics to be Covered

- Software Management Overview: Trends and Definitions
- Software Planning
- Similarities and Differences Between Hardware and Software
- NASA Lessons Learned
- Software Life Cycles
- Software Metrics and Tracking
- Software Perceptions
- Software Management Considerations

Software Management Overview

Objectives



- To discuss some trends in software within NASA.
- To distinguish between software management and software engineering.

- To discuss software management planning.
- To provide an overview of types of mission software.

25th Annual Software Engineering Workshop

Trends in NASA Software (1)

- Substantially more powerful flight computers are coming into play.
- Ground and flight software, in concert, is assuming an increasing role in mission capability and performance.
- New system development tools that have emerged over the past five years are providing powerful, unprecedented capabilities for developing software system components.
- Executive directives and the re-shaping of NASA are requiring the acquisition of new and complex institutional systems.
- Software technology has outpaced NASA's management capability for software.

Excerpted from a presentation to the NASA Chief Engineer on April 17, 1997 by George Albright of NASA HQ Office of Space Science

25th Annual Software Engineering Workshop

Trends in NASA Software (2)

- Hardware memory and processors have increased significantly in capacity and speed.
  - For example, 4 KB used to be a lot for on-board memory
  - Mars Pathfinder memory was 128 MB
- Software applications have become larger and more complex.
- More autonomy in the spacecraft and more complex instruments and data processing.
- Much more software, both on-board and in the ground system.
- Software costs can be as high as 25% of project costs.
- Greater use of generalized hardware with customization in the software (and a desire to move toward reusable software components).
- System architecture is an integrated hardware-software design.

Software is an integral and critical part of today's systems.

NASA Software Issues and Concerns

- The insertion of additional, more powerful computers in the mission creates additional, and fundamentally new challenges in system engineering and quality management.
- Many programs and project managers lack the technical background for exercising an equivalent degree of management insight on software as they do on hardware.
- Existing NASA policies and directives on software have so much "wiggle room" that projects can routinely exempt themselves from software management.
- A recent study by the Software Engineering Institute (SEI) revealed that almost 90% of organizations developing software do so by reliance on personal heroics and "hacking".

Excerpted from a presentation to the NASA Chief Engineer on April 17, 1997 by George Albright of NASA HQ Office of Space Science

General Progress in Software

- The ability to reuse software has increased.
- Models of software have been developed.
- Process models exist and provide guidance for developing software (CMM type of models).
- Cost per function has dropped appreciably.
- There is much more machine portability and independence.
- Languages and compilers are much more efficient and reliable.
- Understanding of key considerations in producing software efficiently and reliably.
  - processes, infrastructure, tools, estimation, tracking (e.g., earned value techniques)
- Concepts of varying life-cycle models and when to apply them.
- Fault tolerance through expert systems.
- Proven techniques of design are understood.
  - (e.g., information hiding, object oriented design, etc.)

Basic Definitions

- **Software:**
  - Programs, procedures, rules, and any associated documentation pertaining to the operation of a computer system [ANSI/IEEE Standard 729-1983].
- **Software Engineering:**
  - The systematic approach to the development, operation, maintenance, and retirement of software through the use of suitable standards, methods, tools, and procedures [ANSI/IEEE Standard 729-1983].
    - **Encompasses many diverse activities including requirements analysis, architectural and detailed design, implementation (coding, programming), assurance, testing, and maintenance.**
    - **Not another term for programming or coding.**
- **Software Management:**
  - A disciplined approach to the planning, tracking, assessing, and controlling of software product development through the selection and use of specific methods, tools, and procedures [JPL D-2352].

Software Processes (1)

- A ***Process*** includes the methods, steps, and management practices used to produce a product.
- Process examples for software include
  - the life-cycle used to develop the software
  - activities within the life-cycle, such as use of peer reviews, code walkthroughs or inspections
- Selection of processes is based on the type of software and its criticality, and schedule/budget constraints.
  - and also its quality goals - (See slides #64-65).
- Organizations that adopt disciplined processes find that:
  - the quality of the software improves.
  - predictability and manageability of development increases, e.g., availability of metrics enables projects to develop more accurate cost and schedule estimates for software.

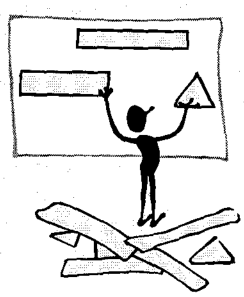
Software Processes (2)

- Process requirements, frameworks, and guidelines include:
  - Capability Maturity Model (CMM) and Key Process Areas (KPAs)*
  - Personal Software Process/Team Software Process (PSP/TSP)*
  - ISO 9000, ISO 9000-3
  - IEEE and ISO Standards and Handbooks
  - For example, specific JPL Processes include:
    - *The JPL Software Development Process Description,* JPL D-15378
    - *Software Development Principles for Flight Systems*, Release 1
    - *JPL Software Management Standards Package*, Rev. 3 JPL D-4000 (for document formats).
- A key management role is to define and enforce the adherence to the selected processes.

* developed by the Software Engineering Institute (SEI) at Carnegie Mellon University (CMU)

Software Planning

Setting the Direction



What is Software Planning?

- Planning:
  - Establishes the framework and benchmark for team members
    - Sets assumptions for carrying out resource estimation
    - Communicates assumed project scope
  - Sets rules of communications and reviewing
  - Defines roles and responsibilities
  - Identifies goals and expectations of the project
    - As well as limitations and assumptions
  - Defines processes to be used in executing the plan
- Planning should not:
  - Trivialize the nature of the software development process (e.g., "small matter of programming") or assume all will go well
  - Assume resource expenditure equates to progress (Brooks' Law)

Typical SMP Topics

- A good Software Management Plan (SMP) should address the following:
  - Key Software Roles and Responsibilities, Staffing Profile
  - System Development Approach
    - Methodology, Life Cycle, Deliverable Products, Reviews
    - Development Standards and Tools
  - Risk Management Approach (if not already covered in PIP)
  - Application Standards, Data Standards & Technology Standards
  - Documentation Plan and Procedures
  - Configuration Management Plan
  - Quality Assurance Plan
  - Security Approach
  - Acronyms and Abbreviations

It is imperative that the plan be established and continually updated as additional information becomes available.

Reasons Software Planning Can Go Awry

- Techniques for estimating software are poorly developed.
  - They reflect the untrue assumption that all will go well.
  - They fallaciously confuse effort with progress.
- Because of uncertainty in software estimates, software managers lack the courteous stubbornness of the 'Chef at Antoine's'.
  - "Good cooking takes time. If you are made to wait, it is to serve you better, and to please you." (menu at Antoine's)
- Schedule progress is poorly monitored.
- When schedule slippage is recognized, the usual response is to add manpower or to compress later phases.

Accumulated experience has enabled us to address some of these historical planning issues.

Excerpted from The Mythical Man-Month by Frederick P. Brooks

Support Activities That Must be Assigned

- 'Process Engineering' activities
  - Define and deploy appropriate processes for the project
    - standards, policies, procedures
  - Carry out measurement activities (e.g. metrics, EVM, etc.)
  - Implement improvement initiatives
  - Independent benchmarking verification (CMM, ISO,...)
- 'Configuration Management' activities
  - Maintain list of all controlled items (require process for change)
  - Establish procedures for executing change to agreed artifacts
- 'Software Quality Assurance' activities
  - Verify agreed processes are utilized
  - Verify each product conforms to agreed form and format
  - Facilitate the deployment of concepts and approaches for quality to the entire project

Software Inventory

- Good software management means that the manager understands the full scope of the software to be managed and developed.
  - How many systems and subsystems?
  - What types of software?
- A software inventory includes a comprehensive list of:
  - all the software that will be developed or procured by the project, as well as
  - its type, criticality, and quality goals.
- The software inventory provides the basis for defining the software processes and plans.

A Project involves multiple, inter-dependent software development efforts, some occurring in parallel.

Types of Project Software

- Onboard Flight Software, including operating systems
- Ground Operations Software (uplink, downlink)
- Mission Planning and Scheduling Software
- Scientific/Analysis Software
- Software Development Environment, including tools for problem management, configuration management, Computer Aided Software Engineering (CASE), etc.
- Software Test Environment including automated test tools, test drivers, debuggers, diagnostic software, test data or databases
- Simulation Models
- System/Project Testbeds, Simulators, etc.
- Hardware Design (CAD) and Diagnostic Software
- Firmware -- FPGAs (Field Programmable Gate Arrays) and ASICs (Application Specific Integrated Circuits)

25th Annual Software Engineering Workshop

Software Criticality Classes

- Just as for hardware, there are Software Criticality Classes (e.g., JPL D-15378, *The JPL Software Development Process Description*):
  - **A**: Mission Critical
  - **B**: Mission Support
  - **C**: Development Support
  - **D**: Non-Deliverable
- Within a software type, there may be different criticalities
  - e.g., compilers and some ground software are "mission critical"
- Meeting software criticality goals is accomplished, not only through the architecture/design, but also through the processes used to develop the software.

25th Annual Software Engineering Workshop

Lessons for Planning

1. Identify Processes Explicitly in Software Planning
2. Demand Quality
3. Plan at least 35% for Testing
4. Identify and Mitigate Risks
5. Capitalize on Previous Experience and Lessons Learned

25th Annual Software Engineering Workshop

Planning Lesson #1

Identify Processes Explicitly

- Planning must include agreed 'approach' for software
  - Approach include standards, policies, procedures
  - Agreement includes management, assurance support, and developers
  - Role of assurance is to record agreed processes (form a contract) then to periodically audit use of processes
- Software team selects processes
  - Within guidelines of the organization, developers (managers) select approach - forms a 'contract'
  - When processes are stipulated top-down, occasional resistance occurs
- Agreement enhances communications and clarifies expectations

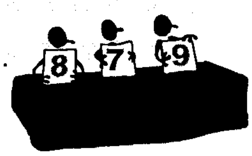
> **✦ Insist that software processes be identified**
> **✦ Assign responsibility for identifying, reviewing, reporting use**

25th Annual Software Engineering Workshop

Planning Lesson #2

Demand Quality

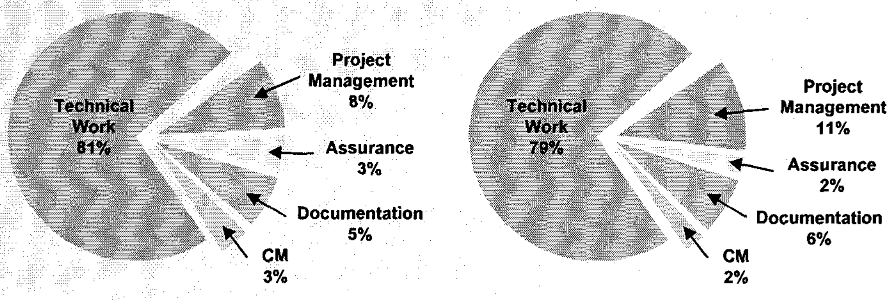
- Establish models of quality
  - Define defect rates as targets - adopt and communicate models
  - Apply measurement to track quality and to raise awareness
  - As measurements (metrics) are analyzed, improved models are built
- Expect quality
  - High-quality software can and should be produced.
    - Defects will always be injected, but techniques exist to remove them.



  - Roles of software manager, assurance support, and process support are to produce high-quality; hold them accountable in planning.
  - Appropriate resources must be allocated to ensure quality
- Allocate approximately 3% to 6% of budget for quality assurance, process engineering, configuration management.
  - Assign specific responsibilities for assurance, process engineering, configuration management.

25th Annual Software Engineering Workshop

Where is Software Effort Spent?



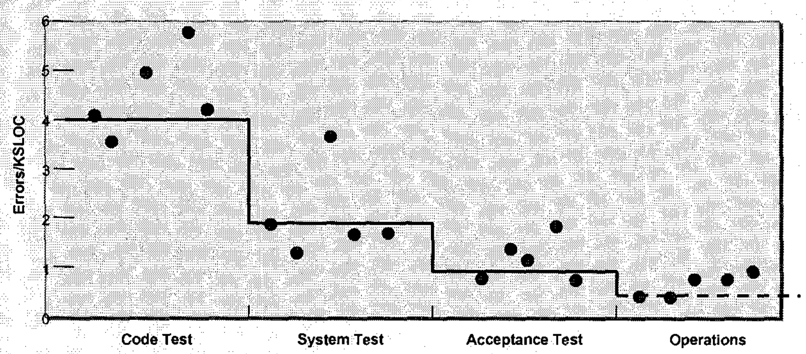
Mature organizations allocate up to 6% of system cost for assurance, process, CM

*** Process Engineering averages 0.9%, but is charged to overall organization (not project)**

25th Annual Software Engineering Workshop

Error Detection Rate

Example of Model Building at NASA

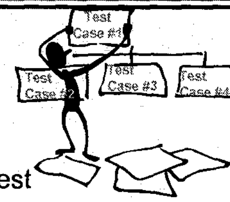


Empirical models aid in planning (e.g., testing) and in performance analysis

* Based on 5 similar projects in NASA/GSFC (ground support systems)

Planning Lesson #3

Plan for Testing

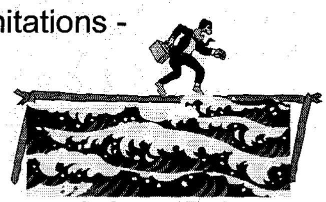


- NASA/GSFC historical data indicates 30-35% ($) for software testing in successful projects
  - Includes system test, integration test, acceptance test
  - Historical information can help determine when to stop testing
- "Failure to allow enough time for software test, is peculiarly disastrous. Since the delay comes at the end of the schedule, no one is aware of schedule trouble until almost the delivery date." <u>The Mythical Man-Month</u>, by *Frederick P. Brooks*
- Testing software is the only means available to determine levels of confidence in performance.
  - Testing will uncover errors, can demonstrate functions are performing correctly, and can provide good data on reliability.
  - But: Testing <u>cannot</u> show or guarantee the absence of defects; it can only show that defects are present.

Planning Lesson #4

Identify and Mitigate Risks

- Software planning has numerous limitations - address them
  - Identify assumptions
  - Plan to accommodate change
    - Requirements, environment, priorities will change



Software Risks

- Identify software risks and develop risk mitigation plans
  - To enable management and control, identify, quantify, monitor and mitigate potential software risks throughout the life-cycle
    - e.g., create a "Red-Yellow-Green" chart to monitor risk status
  - Schedule analysis and reporting milestones
- Set reasonable goals of performance
  - Utilize empirical data (histories) to set goals.

25th Annual Software Engineering Workshop

Planning Lesson #5

Capitalize on Experience

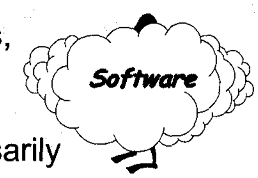
- Every software project is unique, but...
  - We have learned to 'reuse' with caution.
  - Domains and classes of software can provide guidance.
- Historical records of successes and failures should be "required reading".
  - During planning, assure that software staff utilizes historical information (lessons learned).
  - Risk management can be guided by historical perspective.
- Software historical metrics provide:
  - Estimates of units of output per staff size (productivity)
  - Phasing distribution (requirements, design, code, test, QA, CM,..)
  - Models of development

25th Annual Software Engineering Workshop

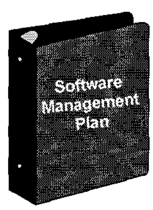
Software Planning Summary

- While there are no "silver bullets", there are proven techniques to manage software.

- Software Management needs to be approached as the technical discipline it is, not as a "mysterious art".



- While software development is not necessarily "harder" or more difficult than hardware development, it IS different and needs to be managed accordingly.
  - A good software management plan and meaningful tracking of progress can go a long way towards mitigating risk and ensuring success.



Similarities and Differences

Objectives

- To explore some similarities and differences between hardware and software

- To understand some unique aspects of the software development process and ways to mitigate them.

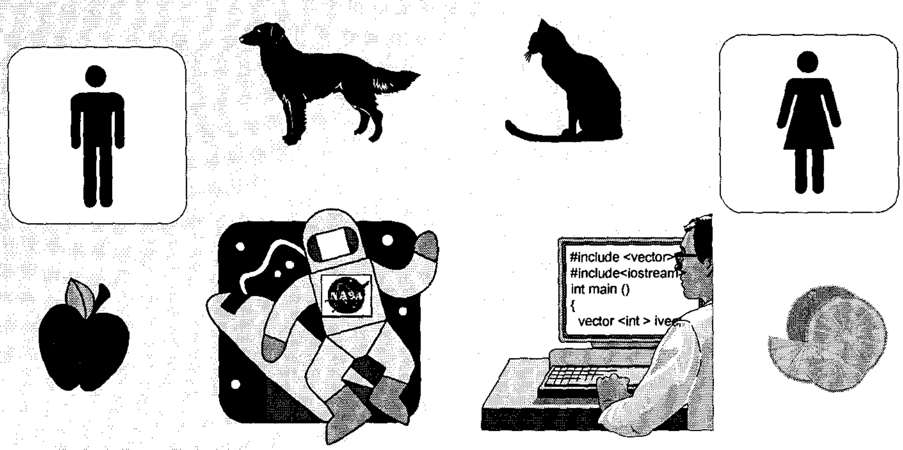
25th Annual Software Engineering Workshop

Similarities of Hardware and Software

- Many similar types of reviews, processes, and deliverable products are required for both hardware and software:
  - Requirements documents and requirements reviews
  - Interfaces need to be carefully defined (ICDs, SISs)
  - Design documents and design reviews
  - Allocation of functionality to components
  - Many COTS products are available at various levels
    - Hardware: components/chips, boards, product, systems
    - Software: functions, programs, libraries, applications, systems
  - Implementation and various levels of testing and test plans
  - User's Guide and/or Operator's Manual
  - Configuration Management and Problem Management
  - Quality Assurance

Differences

- But software is also <u>different</u> from hardware in many ways.



Software is Different from Hardware (1)

Differences with respect to system reliability

- Difference --> Implications --> Mitigation Strategy
- Software does not degrade due to wear or fatigue.
  - But at some point, software can't handle changes in the environment (e.g., new version of the OS, DBMS, or compiler or new inputs, interfaces, hardware speed, or database fills up, etc.
  - Mitigation Strategy: Allocate resources and staff for software maintenance; plan and determine the timing of system upgrades.
- Software results are unconstrained by any laws of physics.
  - If a program expresses the force of gravity with the wrong sign, the calculations will produce the effect of anti-gravity.
  - Mitigation Strategy: Ensure that domain experts review algorithms, computations, and test results for correctness and accuracy.

Excerpted from a "Integrating Software Engineering and System Engineering" by Dr. Barry Boehm, USC Computer Science Dept., The Journal of NCOSE Vol. 1 1994

Software is Different from Hardware (2)

Differences with respect to system reliability (Cont.)

- Software interfaces are conceptual rather than physical.
  - There is no easy-to-visualize three-prong plug and its mate.
  - Mitigation Strategy: Ensure that all external and internal interfaces are thoroughly documented, reviewed and tested to an appropriate level of granularity.
- There are many more distinct logic paths to check in software than in hardware. There are many more distinct entities to check.
  - Any item in a large file may be a source of error.
  - Mitigation Strategies: Employ test coverage software and/or develop and test functional threads through the entire software. Develop and conduct a sufficient number of test cases. Guard and preserve the schedule time for the test period.

Excerpted from a "Integrating Software Engineering and System Engineering" by Dr. Barry Boehm, USC Computer Science Dept., The Journal of NCOSE Vol. 1 1994

Software is Different from Hardware (3)

Differences with respect to system reliability (Cont.)

- The error modes are generally different.
  - Software errors generally come with no advance warning, provide no period of graceful degradation, and more often provide no announcement of their occurrence.
  - Mitigation Strategy: Encourage developers to include error checking code in their programs and corresponding error messages which are truly meaningful. Ensure that your testing team has people who are good at troubleshooting.
- Repair of a hardware fault generally restores the system to its previous condition.
  - Repair of a software fault does not.
  - Mitigation Strategy: Realize that much software maintenance is for the purpose of fixing bugs (the process itself can also introduce new bugs!) and for adding new features. Hence, the maintenance process needs to be managed like the development activity it is.

Excerpted from a "Integrating Software Engineering and System Engineering" by Dr. Barry Boehm, USC Computer Science Dept., The Journal of NCOSE Vol. 1 1994

Software is Different from Hardware (4)

- Software tends to resist freezing. It is truly **soft**.
  - New releases can often require reaching back into previously released components in order to install "hooks" and features required by the new release.
  - Mitigation Strategy: Realize that software's flexibility is a mixed blessing. Ensure that adequate change control and CM are employed, enforce "freeze dates" and ensure that adequate time is allocated for making and testing software changes.
- Software sports an unfamiliar, obtuse jargon.
  - "The boot code pointer array requires a low order bit shift."
  - Mitigation Strategy: Pick a good information modeling approach and then train developers, reviewers and key managers on the methodologies, information modeling approaches and notations used on the project. Train managers in "computer literacy."

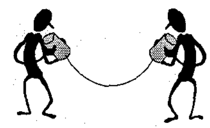
Excerpted from a presentation to the NASA Chief Engineer on April 17, 1997 by George Albright of NASA HQ Office of Space Science

Software is Different from Hardware (5)

- Software testing can be especially challenging.
  - Major system simulators must often be developed, which will usually be daunting system development on their own.
  - Mitigation Strategy: Allocate sufficient schedule time and resources to set up and deploy the test environment, to develop test scripts and test software, and to conduct the test cases and review/analyze the results.
    - **Excerpted from a presentation to the NASA Chief Engineer on April 17, 1997 by George Albright of NASA HQ Office of Space Science**
- Progress in software is slower than in hardware.
  - ". . . the anomaly is <u>not</u> that software progress is slow, but that computer hardware progress is so fast. No other technology since civilization began has seen six orders of magnitude in performance-price gain in 30 years."
    - **Excerpted from "No Silver Bullet: Essence and Accidents of Software Engineering" by Frederick P. Brooks, Jr., Computer Magazine, April 1987**

Essence of Modern Software

- Complexity
  - No two parts of a software entity are alike. . . Software systems have orders-of-magnitude more states than computers do. Complexity increases non-linearly with size.
  - From the complexity comes the difficulty of communication among team members.



- Conformity
  - Much complexity is arbitrary complexity forced by the many human institutions and systems to which software interfaces must conform.
- Changeability
  - Constantly subject to pressures for change since the software of a system embodies its function. . . because software <u>can</u> be changed more easily.
- Invisibility
  - Software is invisible and unvisualizable.

Excerpted from "No Silver Bullet: Essence and Accidents of Software Engineering" by Frederick P. Brooks, Jr., Computer Magazine, April 1987

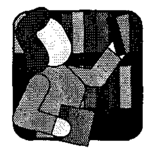
25th Annual Software
Engineering Workshop

NASA Software Lessons Learned

Lessons Learned

- "Those who cannot remember the past are condemned to repeat it." or "Those who do not learn from history are doomed to repeat it." George Santayana, Harvard philosopher

- "We learn from history that we do not learn from history." Georg W.F. Hegel, German philosopher
- Somehow we never believe that what happened to "them" will happen to us or our project.
  - . . . But that attitude leads to complacency.
- Some companies require their managers to spend one to two weeks prior to the start of a project reviewing lessons learned from previous projects.



NASA Lessons Learned

- Of the 900 lessons learned in the NASA Lessons Learned Information System (LLIS) http://llis.nasa.gov/ ,117 are related to software.
- 45 JPL lessons cover topics such as:
  - Specific software problems on missions such as Voyager, Galileo, Mars Observer, Magellan, Mars Global Surveyor, Mars Pathfinder, Mars Climate Orbiter, Mars Polar Lander, etc.
  - Software processes, reviews, methods, fault tolerance and fault protection, real-time operations, IV&V, PFRs, testing, etc.
- 72 lessons from other Centers cover topics such as:
  - Software upgrades, testing, management and planning, ground support equipment (GSE), redundancy, reliability, IV&V, problem reporting, fault tree analysis, maintainability, etc.

25th Annual Software Engineering Workshop

Recent NASA JPL Failures (1)

- Mars Climate Orbiter (MCO) Failure
  - **"Small Forces" software** which created Angular Momentum Desaturation (AMD) file used in trajectory models provided thruster performance data in English units vs. Metric units.
  - No flowdown of **requirements** from higher-level MCO SIS to Software Requirements Document.
  - Lack of rigor in software **specification** -- Small Forces software specification did not state required engineering units.
  - End-user (navigation) representative not specifically requested to attend major **reviews**, walkthroughs or acceptance test.
  - Software **walkthrough** process was not adequate -- required persons not in attendance, SIS not used, action items not published
  - Interface between Small Forces software and Navigation software not **tested**.
  - Inadequate **training** and **staffing**.

25th Annual Software Engineering Workshop

Recent NASA JPL Failures (2)

- Mars Polar Lander (MPL) Failure
  - **Premature Descent Engine Shutdown** -- Touchdown sensors generated a false momentary signal at leg deployment and the flight software (Touchdown Monitor) incorrectly interpreted these spurious signals as valid touchdown events. When the sensor data was enabled at an altitude of 40 meters, the engines would shutdown and the lander would probably free fall to the surface.
  - Incomplete **requirements** specification and flowdown regarding failure modes for leg deployment and touchdown led to software **design** that did not properly account for the presence of transient or spurious signals.
  - Software **test** planning did not identify all sets of conditions that could "break" the software.

Recent NASA JPL Failures (3)

- Mars Global Surveyor (MGS) Failure
  - **Aerobraking Extra Burn Anomaly (1997)** -- An unintended repetition of MGS Aerobraking Maneuver #5 occurred after the intentional burn, i.e, the spacecraft unexpectedly repeated the aerobraking maneuver, imparting an additional delta-v to the S/C.
    - To avoid a deeper penetration into the Martian atmosphere, the flight team designed, uploaded, and successfully commanded a third "antidote" maneuver which counteracted the effect of the unplanned second maneuver.
  - The cause of the incident was traced to an **incompatibility** between ground and flight software. A new version of the ground software had been installed in the ground data system one day earlier than planned. This new software had been redesigned to improve programming and command efficiency. Certain memory partitions were reassigned in on-board memory.
    - Due to the address changes, the recently executed Aerobraking Maneuver #5 was loaded into the partition originally intended for the next science sequence. Hence, when the spacecraft command data subsystem (CDS) executed "Next Sequence," the Aerobraking Maneuver #5 sequence was still present in memory, and it was performed instead of the science sequence.

Recent NASA JPL Failures (4)

- Mars Global Surveyor (MGS) Failure (Cont.)
  - **Configuration Management**
    - Prior to implementing any change in command and control software, verify that the configurations of ground and flight software are compatible.
    - Provide a method for identifying the time-criticality of any flight software configuration change.
  - Software **design** and **implementation** should minimize the possibility of accessing onboard stored command sequences at inappropriate times.
  - Development and use (**operations**) of flight and ground software should be closely coordinated from the very beginning and throughout the life of the project.

25th Annual Software Engineering Workshop

Causes of Software Cost Growth at NASA JPL

Risk Area	Cost Growth Sources	Cost Growth Causes		
		Process	People/Teams	Tools & Methods
Planning	• Poor planning and estimation practices • Insufficient reserves for SW	• No generally accepted planning process for software development.; planning is largely dependent on the individual engineer (preparing the plan) • Uniqueness of software not captured in initial stages (functional to deliverable) • SW requirements and design are more volatile & solidify later than hardware in the life cycle. • Don't know how to freeze software requirements the same way we know how to freeze hardware requirements	• SW team not included in early stages of planning • SW not recognized in initial planning	• Poor and constantly changing assumptions and cost estimation methods • Lack of software planning tools • Lack of SW cost metrics.
Require-ments & Design	• Lack of good architecture and system partitioning • Systems decisions made without accounting for impact on software	• Subsystem view of spacecraft -- not viewed as important to have a top-level architecture early in the project. • Software design is traditionally done at the subsystem level (based on hardware perspective) • Architectural issues are not sufficiently worked out in Phase A/B • Concurrent development can lead to interface problems due to lack of communication between teams especially when there is schedule compression.	• No awareness or recognition even at the mission & system level that software needs to be addressed. • Don't view architecture as a software intensive process	
Experience & Teaming	• Insufficient software experience among managers and system engineers • Poor teaming between HW/ SW and systems/SW team		• Management and system engineers have limited SW experience • Engineers grew up in a hardware intensive world. • Managers and system engineers do not view software engineers as broad enough. • Lack of software-system engineers • Software culture is underdeveloped at the present	
Testing	• Testbeds; too few, too late, not validated, insufficient capability • Lack of early test planning; lack of functionality,	• Lack of sufficient funding. • Testbeds not listed in WBS; not accountable. • Lack of sufficient schedule or recognition of the importance of testing. • "Big Bang" style testing waits until end to test. • Test documents not in place until late in life cycle	• Lack of education & appreciation of value for testbeds. • Test team not in place until late in life cycle • Integration and SW teams not available to support ATLO	• Dependence on hardware testbeds. • Lack of tools and under utilization of existing tools • Lack of controlled tests and test data
Software Inheritance	• Inherited code did not behave as advertised, was poorly documented, and required more modification than expected	• Lack of software inheritance review process. • Inheritance not distinguished between reusable code and code that has not been designed for that purpose. Inheritance (typically) only reuses the design. • No incentives for projects to develop fully reusable code.	• Many projects fail to bring onboard the original developers when they attempt to inherit software	• Too many advantages of inheritances assumed, esp. cost savings • Cost models don't properly account for COTS, sw inheritance and modification.. • Too often assumed that COTS costs are free

Ref: JPL D-18660, "Flight Software Cost Growth: Causes and Recommendations"

25th Annual Software Engineering Workshop

NASA JPL Recommendations by Risk Area

Risk Area	Cost Growth Sources	Recommendations		
		Process	People/Teams	Tools/Methods
Planning, Estimation & Control	• Poor planning and estimation practices • Insufficient reserves for SW	1. Need a focused end point with clear success criteria 2. Need better tailored risk management plan with appropriate contingencies 3. Allocate larger percentage reserves to software		
Require-ments & Design	• Lack of good architecture and system partitioning • Systems decisions made without accounting for impact on software	4. Require that a clear understanding of SW be included as part of NAR approval 5. Need good architecture to define demarcation between HW and SW	6. System Engineers need to understand that the software provides the system level interfaces 7. Do not look at SW as separate item but see as part of an integrated system design	
Experience & Teaming	• Insufficient software experience among Managers and system engineers. • Poor teaming between HW/ SW and systems/SW team		8. Project office needs to have some SW expertise 9. SW team needs to understand system[1] 10. Everyone should have some mission level training to provide end-to-end understanding of the system [1]	
Testing	• Testbeds; too few, too late, not validated, lacked capability • Lack of early test planning; lack of functionality,	11. Testbeds and simulators need to be made a major product deliverable that is completed early in lifecycle	12. Need to have a dedicated integration team and a dedicated test team whose job is it to break the software 13. Require a test engineer be a member of the early planning team and reviews.	
Software Inheritance	• Inherited code did not behave as advertised, was poorly documented, and required more modification than expected	14. Need a software inheritance review	15. For Inheritance people need to come with the software	16. To increase the amount of Inheritance between projects, need to create infrastructure to provide incentives to develop reusable code and to maintain it.

Ref: JPL D-18660, "Flight Software Cost Growth: Causes and Recommendations"

25th Annual Software Engineering Workshop

Software Capability Maturity Model (SW-CMM)

25th Annual Software Engineering Workshop

Software Capability Maturity Model (1)

Characteristics of CMM Levels

CMM is a yardstick for measuring ability to deliver a quality product on schedule and within budget

Level 5 "Optimizing" – Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

Level 4 "Managed" – Detailed measures of software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.

Level 3 "Defined" – The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.

Level 2 "Repeatable" – Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on project with similar applications.

Level 1 "Initial" – Software process is characterized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends on individual effort. Usually involves "heroics".

CMM was developed by the Software Engineering Institute (SEI), Carnegie Mellon University (CMU)

Software Capability Maturity Model (2)

CMM Levels and corresponding Key Process Areas (KPAs)

Level 5 "Optimizing"
- Process Change Management
- Technology Change Management
- Defect Prevention

Level 4 "Managed"
- Software Quality Management
- Quantitative Process Management

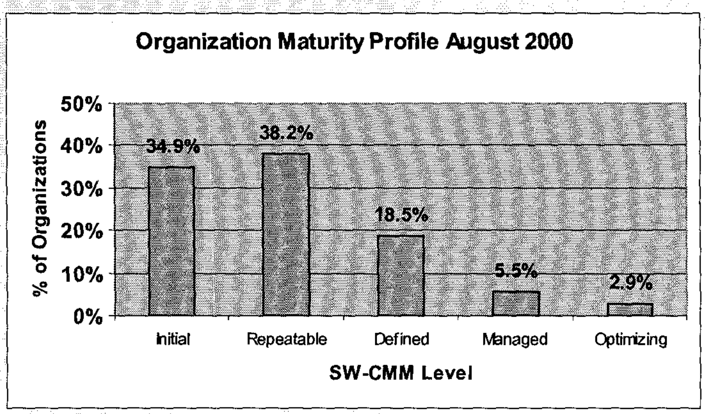
Level 3 "Defined"
- Software Product Engineering
- Intergroup Coordination
- Software Process Definition
- Integrated Software Management
- Peer Review
- Training Program
- Software Process Focus

Level 2 "Repeatable"
- Software Configuration Management
- Software Subcontract Management
- Requirements Management
- Software Quality Assurance
- Software Project Planning
- Software Project Tracking & Oversight

Level 1 "Initial"
- Few processes defined
- Usually involves "heroics"

Excerpted from A Discipline for Software Engineering by Watts S. Humphrey

Organization Maturity Profile



Based on assessments conducted from 1996 through June 2000, of 901 organizations, 276 participating companies, 4174 projects.

http://www.sei.cmu.edu/sema/profile.html

SW-CMM for NASA Centers

- Dan Goldin, the NASA Administrator, in testimony to the Congressional Committee on Science on 06/20/2000, stated that "all NASA Centers are developing plans for attainment of CMM Level 3 (Defined) for organizations producing critical software."
  - The NASA CIO has chartered the NASA Software Working Group (SWG) to prepare the plan for improving NASA software and attaining SW-CMM Level 3.
- Consider steps that you, as a Project Manager, could take to address SW-CMM Level 2 Key Process Areas (KPAs) on your project such as:
  - Software Project Planning
  - Software Project Tracking and Oversight
  - Software Subcontract Management
  - Software Requirements Management
  - Software Configuration Management
  - Software Quality Assurance

Software Life-Cycles

Objectives:

- Show the role of software life-cycles in defining software development activities
- Provide some criteria for selecting life-cycles and example life-cycles and how they fit into a project life-cycle
- Show impact of some key software life-cycle issues:
  - prototyping
  - life-cycle products (documentation)
  - maintenance

What Is A Software Life-Cycle ?

Software Life-cycle provides a framework for:

- Defining and grouping sub-processes (activities)
  - the major engineering activity provides the name of the "phase" (but don't forget other important activities)
  - management/tracking/reporting; engineering; test/assurance
- Showing completion and coordination points with other project elements or other sub-processes (reviews)
  - Plan/Commitment, Software Requirements, Software Design, and Operational Readiness Reviews
- Defining and maintaining controlled records (products)
- High-level structuring of risk mitigation strategies

Most projects use iterative life-cycles, but all show some form of "requirements-design-build-test."

Most life-cycles are frequently realized as GANTT-type schedules.

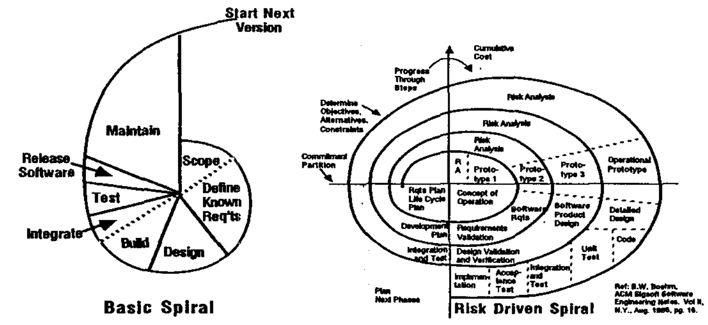
Iterative Life-Cycles

- Iterative models are better able to cope with:
  - Early delivery(ies) required to demonstrate functionality
  - Poorly understood and/or changing requirements or environment
  - Early support of integration and test activities
  - Early user/operator feedback
  - Staffing limitations
- But can have drawbacks:
  - Additional rework between iterations (rule of thumb 20%, but less than for waterfall)
  - More complex software management, CM, baselining, reviews, deliveries to operations

25th Annual Software Engineering Workshop

Iterative Examples

Generic Iterative Forms



Allow iterating through some or all parts of a development cycle.

25th Annual Software Engineering Workshop

Rapid Prototyping

- Generally considered to be a "partial" life-cycle used to "discover" requirements or "explore" implementation options
- Must be careful with respect to products:
  - Products are intended to be thrown away or
  - Must be re-engineered to meet quality requirements
- Can be seen as early (inner) iterations in a spiral; must be followed by a waterfall or iterative development process to produce viable products
- Prototyping is highly effective as an **activity** contained within a phase of an iterative model

Prototyping Activities

- Prototyping is a "highly adapted" development activity; one takes a lot of short-cuts to answer specific questions
- Prototyping supports many of the life-cycle processes:
  - e.g., requirements definition, design trades, system performance, implementation options, technology development; tool selection
- Prototyping must have a plan (goal and schedule) and a product
- By-products may be used in "real" development if brought up to necessary quality standards
- Should NOT be used as a way to avoid doing requirements analysis

Maintenance Activities

- Corrective maintenance: used to overcome faults.
- Perfective maintenance: used to improve performance, maintainability, or other software attributes.
- Adaptive maintenance: used to keep a software product usable in a changed environment.
- Supportive maintenance: used to explain the software, plan for future support, or to measure software attributes.
- Much of maintenance performed is perfective or adaptive.
- All software will need maintenance.
- Maintenance <u>must</u> be included in planning.
  - What and how much depends on the use of the software.

Tying the Software Life-Cycle to the Project Life-Cycle (1)

- How many developments does the Project have?
  - think of the different kinds of software you identified in your software inventory
  - each of those has its own development (or procurement) cycle
  - each must be tied to other related software developments and to the project life-cycle
- Management is responsible for coordinating these "interfaces"

>>> Complexity is the villain! <<<

Tying the Software Life-Cycle to the Project Life-Cycle (2)

Typical JPL Project Life-Cycle

NASA Phases		FORMULATION		IMPLEMENTATION		
JPL Life Cycle Phases	**Pre-Phase A:** Advanced Studies	**Phase A:** Mission & Systems Definition	**Phase B:** Preliminary Design	**Phase C:** Detailed Design	**Phase D:** Build and Test	**Phase E:** Operations
Major JPL Reviews		PMSR	Project PDR	Project CDR	ARR MRR	PLAR CERR

Concept Review ICR Commitment CR 11/22/2000 Contract Launch

Software Life-Cycle Products

- Software life-cycle "products" include planning documents, schedules, reviews, technical documentation, software executables, test scripts, test results, etc.
- Roll-out complex parts/issues to better manage risk
- Package documentation to fit:
  - life-cycle phases
  - timing of baselining and control (CM convenience and ISO)
  - level of risk/complexity
  - authorship
  - level of approval

Example Software Document List

- Software Management Plan (SMP)
- Software Requirements Document (SRD)
- Software Interface Specifications (SIS)
- Software Design Document (SDD)
- Software Integration and Test Plan (SITP-1) - Planning
- Software Integration and Test Plan (SITP-2) - Procedures
- Software Integration and Test Plan (SITP-3) - Reports
- Software User's Guide (UG)
- Software Operator's Manual (SOM)
- Release Description Documents (RDDs)
- Delivery, Installation, Operations, and Maintenance Plan(s)

Software Disciplines and Skills

Just as there are many sub-disciplines in hardware, so there are also many sub-disciplines in software and different skill sets.

- Software Disciplines:
  - real-time and embedded systems
  - transaction systems
  - client-server systems
  - data analysis/reduction/ processing
  - database
  - image processing
  - operating system and tools
  - processes and methods, etc.

Software Skills:
- software managers
- software architects
- developers and programmers
- domain experts
- software testers
- software assurance
- configuration management
- database administrators (DBAs)
- system administration(SAs), etc.

Your project will require a mix of personnel throughout the life-cycle.

Software Measurement

Software Measurement

- What is Software Measurement?
  - Software Measurement is the activity of determining quantifiable attributes of both products and process. It is the 'engineering' element of software.
- Measurement (metrics) can help determine:
  - Are the software development activities on schedule?
  - Is the software reliable?
  - Should I make changes?
  - How long will testing take?
  - Will the software perform all critical functions?

Reasons for Measurement

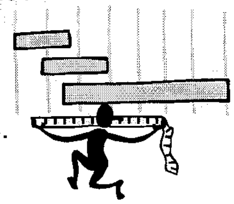
- There are 3 reasons for software measurement:

1. **Assisting Project Management**
   - Validate mission goals
   - Evaluate the quality of the process/product
   - Support decision-making - provide visibility
2. **Establishing Models**
   - Engineering software
   - Create a corporate memory - baselines/models of current practices
3. **Guiding/Demonstrating Change**
   - Develop a rationale for adopting/refining techniques
   - Assess the impact of techniques

25th Annual Software Engineering Workshop

Measuring to Assist Project Management

- Measuring progress
  - Are the software development activities on schedule?
    - Earned Value Management (EVM) is our best and most successful measurement example.
    - Tie EVM to <u>all</u> life-cycle products, <u>not</u> lines of code or only the number of modules.
  - Should I make changes? What types of changes?

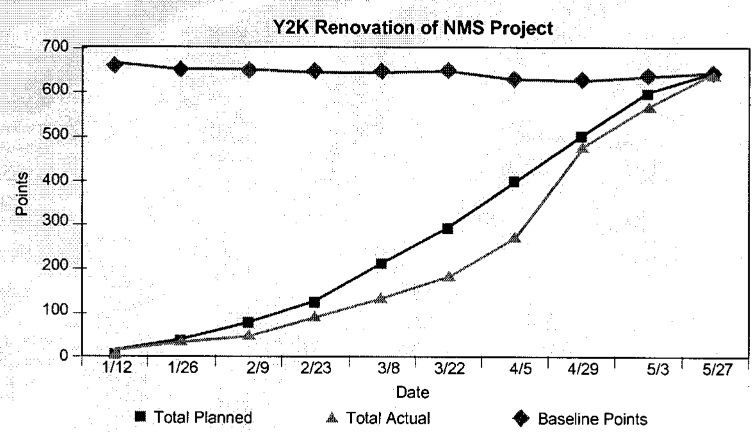


- Measuring quality
  - Will the software perform correctly?
  - Will the software fail? at critical times?
- Measuring functionality
  - Will the software do all the things it is expected to do?
  - Will all capabilities be included?

People don't do what you <u>expect</u>, they do what you <u>inspect</u>.

25th Annual Software Engineering Workshop

Simple Example of Managing Progress



Earned value is an excellent goal utilizing measures for management.

Each module (165) assigned 4 points: 1-designed; 2- coded; 3-inspected; 4-integrated

25th Annual Software Engineering Workshop

Measuring to Guide Change

- Guide changes to your project as well as to your organization
- The most difficult and least used reason
- Success requires mature 'characterizing'
- This is what continuous improvement is all about (such as, CMM).
- Why is it so difficult?
  - Clear description of goals must be stipulated.
  - Sustained effort on Reason 2 (model building)
  - Commitment of planning and resources
  - 'Mature' organization

25th Annual Software Engineering Workshop

Example of Measuring to Guide Change

Question		Process/Technology Change		Goal (Product Measure)
Will	→	Earned Value Techniques	→	Eliminate Surprise
Will	→	Software Inspections	→	Decrease Defect Rates
Will	→	Object-Oriented Technology	→	Increase Reuse Levels
Will	→	Computer-Aided Software Engineering	→	Raise Productivity
Will	→	Independent Verification and Validation	→	Improve System Reliability
Will	→	Spiral Model	→	Decrease Cycle Time
Will	→	SW-CMM Level 3 (or 4 or 5)	→	Decrease Risk

Software Perceptions

Objectives

- To provide information on some perceptions about software that could be insightful to project management.
- To highlight key points in software management.

25th Annual Software Engineering Workshop

Perceptions -- Two Sides of the Software Coin (1)

Views of Project Managers (PMs)	Views of Software Engineers (SWEs)
1. SWEs are not good at estimating costs and schedules.	1. PMs don't accept realistic estimates, but insist on cuts OR they come to us with already approved budgets and schedules ("done deal") that are unrealistic.
2. It's hard to know the true status of the software. I can't tell what's going on "over there"!	2. PMs often don't take the time or know how to ask the right questions to get to the heart of issues OR they ignore our warnings about the effects of tradeoffs.
3. Why does it "take so long" to develop software?! Isn't it "just a small matter of programming"?	3. PMs don't appreciate the difficulty of software or understand the "ripple effect" of changes.

25th Annual Software Engineering Workshop

Perceptions -- Two Sides of the Software Coin (2)

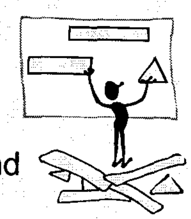
Views of Project Managers (PMs)	Views of Software Engineers (SWEs)
✦ 4. People on my project want to take irrelevant classes and expect me to pay for it.	✦ 4. PMs are unwilling to allocate time and money for training, but SW/IT is a dynamic field and we don't want to become "dinosaurs"!
✦ 5. SWEs are too focused on new technology and playing with more features instead of delivering a product to schedule.	✦ 5. We enjoy technical challenges which is why we chose this field. PMs should establish and enforce realistic "freeze dates" and we'll abide by them. (Line managers find it hard to hire SWEs when the technology on the task is outdated.)
✦ 6. It's hard to find people with the right skills for my project and when I do they don't stay 'til the finish.	✦ 6. Turnover is a fact of life. Not all parts of the life-cycle are as exciting to me nor am I equally skilled in each one.

Software Management Considerations

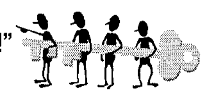
AKA

"Project Manager's Top 13 List"

Software Management Considerations (1 - 4)



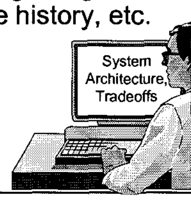
- 1. You need to do accurate software cost estimation and software planning up front!
  - "Those who fail to plan, plan to fail!"
- 2. You need the right skills mix on the team and a capable Software Manager to oversee it all.
  - Software Engineers are not an interchangeable commodity.
- 3. Identify key software risks up front, and then mitigate and manage risks throughout the project life-cycle.
- 4. Make sure that sufficient rigor, methodologies and discipline are applied throughout the software development process.
  - "As a management style, anarchy doesn't scale well!"



25th Annual Software Engineering Workshop

Software Management Considerations (5 - 8)

- 5. Baseline software requirements early, and then manage "scope creep". Ensure software requirements are based on an accurate operations concept.
- 6. While using COTS software can often save development time, its use has implications for flexibility and maintainability.
  - COTS does not equal "turn key".
- 7. Software inheritance needs to be carefully considered.
  - Evaluate compatibility with requirements, underlying design decisions, documentation, performance and failure history, etc.
- 8. Software people need to be involved in developing the system architecture and in making system tradeoffs.



25th Annual Software Engineering Workshop

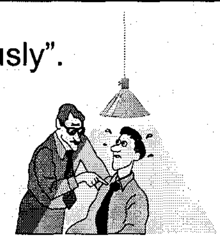
Software Management Considerations (9 - 11)

- 9. Be sure all external and internal software interfaces are adequately documented, signed off, adhered to and tested.
- 10. You must preserve and protect the schedule time allocated to testing and delivery to operations.
  - People will try to push against these deadline boundaries, but you MUST protect this period!
  - Don't just allocate test time. Allocate time to fix bugs and retest.
  - The act of delivering takes time. Work backwards from your delivery date to determine "freeze dates".

- 11. Remember that software quality is built-in throughout the entire development process, not just tested for at the end.

Software Management Considerations (12 - 13)

- 12. Track and communicate status "religiously". Make sure you have enough insight and visibility into the software status to make intelligent decisions.



  - "The Truth Seeker impacts the truth that the Truth Teller tells."
  - Collect and track metrics, monitor PFRs, and use "inch pebbles".
  - Use detailed checklists to ensure that all parts of the task are completed.

- 13. While there are no "silver bullets", there are **proven techniques to manage software**. Avail yourself of this information and/or hire someone who already knows it.

Closing Challenge

- **Good managers leave a positive legacy!**

  - Reliable, maintainable, expandable, well-documented system

  - User-friendly system -- easy to use, easy to operate
  - Enthusiastic, well-trained workforce ready for the next challenge on the horizon

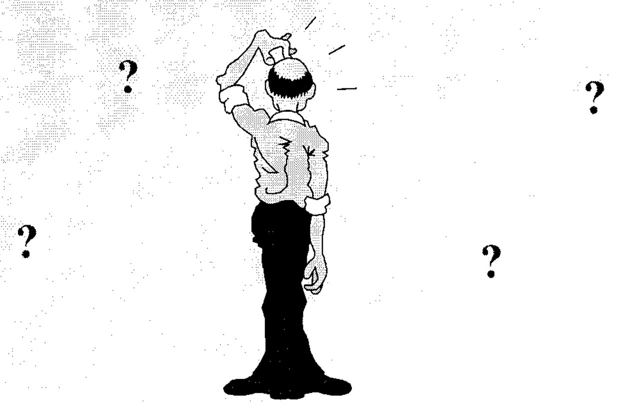
25th Annual Software Engineering Workshop

References (URLs)

- http://www.spmn.com -- Software Program Managers Network; home of 16 best practices and Guidebook for Software Acquisition
- http://www.stsc.hill.af.mil/index.asp -- STSC home page, Lessons Learned and Process information; CrossTalk Journal -- good current articles on software issues
- http://www.sei.cmu.edu/ - Software Engineering Institute (SEI),DOD FFRDC at Carnegie Mellon University contains general process information, COTS and architecture info.
- http://www.software.org/ -- Software Productivity Consortium - consortium of industry and government focusing on software
- http://psmsc.com/ -- Practical Software & Systems Measurement Support Center (PSMSC)
- http://www.computer.org/ IEEE Computer Society site
- http://www.tcse.org/ IEEE Technical Council on Software Engineering (TCSE)
- http://info.acm.org/ ACM site
- http://gantthead.com/Gantthead/default/ New commercial site on managing projects
- http://llis.nasa.gov/ NASA Lessons Learned Information System site
- http://www.ivv.nasa.gov/SWG/index.shtml NASA Software Working Group (SWG) site
- http://www.ivv.nasa.gov/index2.shtml NASA IV&V Center (Fairmont, WV) with links to software sub-groups

25th Annual Software Engineering Workshop

Question and Answer Period



Backup Slides:

Recommendations to Avoid Software Cost Growth by Risk Area

Software Quality

Recommendations for Experience & Teaming

- Need project managers and system engineers who understand software.
- System engineers need to understand that software provides the system level interfaces.
- Project office needs to have some software expertise.
- Need to build a team that can work together and communicate (includes across hardware/software).
- Project Managers need to be able to identify staffing problems early.

All recommendations excerpted from
Flight Software Cost Growth: Causes and Recommendations
(JPL D-18660) February, 2000
by Jairus M. Hihn and Hamid Habib-agahi

Recommendations for Planning

- Need a focused end point with clear success criteria
- Need better tailored risk management with contingency plans
- Need a plan you can track and hang your hat on based on a complete life-cycle
- Software must have an early presence even in pre-Phase A and be part of an integrated plan
- Allocate larger reserves to software
- Require that a clear understanding of software be included as part of NAR approval
- Need more detailed planning and tracking of software similar to hardware
- When putting together a plan, get inputs from everyone and negotiate. Add schedule slack, but make sure all managers know they are accountable.
- Need to change rules of thumb. e.g., software development vs. test used to be 50/50 now appears to be 15/85.

Recommendations for Requirements & Design

- Must have a development process that deals with evolving requirements and assumes things will break.
  - Early and extensive prototyping
  - Incremental deliveries and evolving documents
  - Isolate interfaces
- Identify standardized software functions and put in hardware.
- Need good architecture to define demarcation between hardware and software.
- Do not look at software as separate, but see as an integrated design.
- Get a baseline and configuration management (CM) in place so can carefully manage prioritized requirements.

25th Annual Software Engineering Workshop

Recommendations for Testing & Tools

- Testing
  - Need to have many and varied software test environments.
  - Need to have a dedicated integration and a dedicated test team whose job it is to break the software.
  - Test beds and simulators need to be made a major product deliverable that is completed early in life-cycle.
- Tools
  - Make sure target and development systems are the same.
  - Use design tools with proven record.
  - Get methodology and process in place before purchasing tools.
  - Need good test analysis tools.

25th Annual Software Engineering Workshop

Recommendations for Inheritance & Staffing

- Inheritance
  - Need a software inheritance review.
  - For successful software inheritance, developers need to come with the software.
- Staffing
  - We need to go outside to get more expertise.
  - Software team needs to understand the system.
  - Plan to over staff software engineers to deal with turnover.
  - Need a mechanism to hire more software people without elaborate hiring procedures.
  - Everyone should have some mission level training to provide end-to-end understanding of the system.

Software Quality Goals

- Examples of Quality Goals *:
  - Performance
  - Efficiency
  - Correctness
  - Reliability
  - Reusability
  - Usability
  - Verifiability
  - Portability
  - Maintainability
  - Extensibility

* from SORCE Technical Memorandum 14 (JPL D-10459)

Software Quality

- When defining quality goals, don't just address the major categories of software - onboard, ground, development environment -- but also address the major components within each category.
- Did you consider?
  - The reliability and performance of the onboard operating systems
  - The robustness of the uplink/downlink software
  - The capacity and throughput performance requirements of the science data product generation software
  - The correctness of the compilers in the Software Development Environment
  - The timing and fidelity in the testbeds
  - The accuracy levels of the simulators
  - The usability of development and test tools and related procedures